SimPhoNy Documentation

Release 0.1.1

SimPhoNy, EU FP7 Project (Nr. 604005)

March 20, 2015

Contents

1 Simphony-common **1**
- 1.1 Repository . . . 1
- 1.2 Requirements . . . 1
- 1.3 Installation . . . 2
- 1.4 Testing . . . 2
- 1.5 Documentation . . . 2
- 1.6 Directory structure . . . 2

2 User Manual **3**
- 2.1 Plugins . . . 3

3 API Reference **5**
- 3.1 Core . . . 5
- 3.2 CUDS . . . 5
- 3.3 HDF5 IO . . . 22

4 Indices and tables **29**

Python Module Index **31**

CHAPTER 1

Simphony-common

The native implementation of the SimPhoNy cuds objects and io code (http://www.simphony-project.eu/).

1.1 Repository

Simphony-common is hosted on github: https://github.com/simphony/simphony-common

1.2 Requirements

- enum34 >= 1.0.4
- stevedore >= 1.2.0

1.2.1 Optional requirements

To support the cuba-generate script the following packages need to be installed prior to installing Simphony:

- click >= 3.3
- pyyaml >= 3.11

To support the HDF5 based native IO:

- PyTables >= 3.1.1

To support the documentation built you need the following packages:

- sphinx >= 1.2.3
- sphinxcontrib-napoleon >= 0.2.10
- mock

Note: Packages that depend on the optional features and use setuptools should append the `H5IO` and/or `CUBAGen` identifier next to simphony in their `setup_requires` configuration option. For example:

```
install_requires = ["simphony[H5IO, CUBAGen]"]
```

Will make sure that the requirements of H5IO and CUBAGen support are installed. (see setuptools extras for more information)

1.3 Installation

The package requires python 2.7.x, installation is based on setuptools:

```
# build and install
python setup.py install
```

or:

```
# build for in-place development
python setup.py develop
```

1.4 Testing

To run the full test-suite run:

```
python -m unittest discover
```

1.5 Documentation

To build the documentation in the doc/build directory run:

```
python setup.py build_sphinx
```

Note:

- One can use the –help option with a setup.py command to see all available options.
- The documentation will be saved in the `./build` directory.

1.6 Directory structure

There are four subpackages:

- core – used for common low level classes and utility code
- cuds – to hold all the native cuds implementations
- io – to hold the io specific code
- bench – holds basic benchmarking code
- examples – holds SimPhoNy example code
- doc – Documentation related files
  - source – Sphinx rst source files
  - build – Documentation build directory, if documentation has been generated using the `make` script in the `doc` directory.

CHAPTER 2

User Manual

2.1 Plugins

The SimPhoNy library can extended through two entry points for contributing python modules that container engine and visualisation components:

- `simphony.engine` – A python module that provides one or more classes that implement the `ABCModelingEngine` interface.
- `simphony.visualisation` – A python module that provides a simple function to show (visualise the high level CUDS containers)

To declare that a package contains a visualisation or engine modules for simphony, a developer has to add an entry point definition in the `setup.py` of the contributing package.

e.g.:

```
setup(
   entry_points={
    'simphony.engine': ['<name> = <module_path>'])
```

Where `<module_path>` is a module where the engine class(es) can be found like `my_cool_engine_plugin.cool_engine342` and `<name>` is the user visible name that the `cool_engine432` module will have inside the SimPhoNy framework. It is important that <name> is unique and specific to the contributed components (e.g. name == 'default' is probably a very bad choice)

e.g.:

```
setup(
   entry_points={
    'simphony.engine': ['cool = my_cool_engine_plugin.cool_engine342'])
```

Will allow the user to import the new engine from inside the `simphony` module as follows

```
from simphony.engine import cool
# cool is now a reference to the external module ``my_cool_engine_plugin.cool_engine342``
# If the name of the provided engine class is EngFast then the user should be able to do
engine = cool.EngFast()
```

Note: The `examples/plugin` folder of the simphony-common repository contains a dummy package that contributes python modules to both

CHAPTER 3

API Reference

3.1 Core

Core components and objects of the simphony package.

Classes

DataContainer(*args, **kwargs)	A DataContainer instance

Implementation

class simphony.core.data_container.**DataContainer**(**args*, ***kwargs*)

Bases: dict

A DataContainer instance

The DataContainer object is implemented as a python dictionary whose keys are restricted to be members of the CUBA enum class.

The data container can be initialized like a typical python dict using the mapping and iterables where the keys are CUBA enum members.

For convenience keywords can be passed as capitalized CUBA enum members:

```
>>> DataContainer(ACCELERATION=234)  # CUBA.ACCELERATION is 22
{<CUBA.ACCELERATION: 22>: 234}
```

__setitem__(*key*, *value*)

Set/Update the key value only when the key is a CUBA key.

update(**args*, ***kwargs*)

3.2 CUDS

3.2.1 Abstract CUDS interfaces

Containers

ABCMesh	Abstract base class for mesh.
abstractparticles.ABCParticleContainer	
ABCLattice	Abstract base class for a lattice.

Description

class simphony.cuds.abstractmesh.**ABCMesh**
Abstract base class for mesh.

name
str

name of mesh

add_cell(*cell*)

add_edge(*edge*)

add_face(*face*)

add_point(*point*)

get_cell(*uid*)

get_edge(*uid*)

get_face(*uid*)

get_point(*uid*)

has_cells()

has_edges()

has_faces()

iter_cells(*cell_ids=[]*)

iter_edges(*edge_ids=[]*)

iter_faces(*face_ids=[]*)

iter_points(*point_ids=[]*)

update_cell(*cell*)

update_edge(*edge*)

update_face(*face*)

update_point(*point*)

class simphony.cuds.abstractparticles.**ABCParticles**
Abstract base class for a Particles item.

name
str

name of particles item.

add_bond(*new_bond*)

add_particle(*new_particle*)

get_bond(*uid*)

get_particle(*uid*)

has_bond(*uid*)

has_particle(*uid*)

iter_bonds(*uids=None*)

iter_particles(*uids=None*)

remove_bond(*uid*)

remove_particle(*uid*)

update_bond(*bond*)

update_particle(*particle*)

class simphony.cuds.abstractlattice.**ABCLattice**
Abstract base class for a lattice.

name
str

name of lattice

type
str

type of lattice

base_vect
D x float

base vector of lattice

size
tuple of D x int

lattice dimensions

origin
D x float

lattice origin

get_coordinate(*index*)
Get coordinate of the given index coordinate.

Parameters **index** (*D x int*) – node index coordinate

Returns **coordinates**

Return type D x float

get_node(*index*)
Get the lattice node corresponding to the given index.

Parameters **index** (*tuple of D x int*) – node index coordinate

Returns **node**

Return type *LatticeNode*

iter_nodes(*indices=None*)
Get an iterator over the LatticeNodes described by the indices.

Parameters **indices** (*iterable set of D x int, optional*) – node index coordinates

Returns An iterator over LatticeNode objects

Return type iterator

update_node(*lat_node*)
Update the corresponding lattice node.

Parameters **lat_node** (LatticeNode) –

3.2.2 Pure Python implementation

Classes

LatticeNode(index[, data])	A single node of a lattice.
Lattice(name, type, base_vect, size, origin)	A Bravais lattice
Particle([coordinates, uid, data])	Class representing a particle.
Bond(particles[, uid, data])	Class reprensenting a bond.
particles.ParticleContainer	
Mesh(name)	Mesh object to store points and elements.
Point(coordinates[, uid, data])	Coordinates describing a point in the space
Face(points[, uid, data])	Face element
Cell(points[, uid, data])	Cell element

Functions

make_hexagonal_lattice(name, h, size[, origin])	Create and return a 2D hexagonal lattice.
make_square_lattice(name, h, size[, origin])	Create and return a 2D square lattice.
make_rectangular_lattice(name, hs, size[, ...])	Create and return a 2D rectangular lattice.
make_cubic_lattice(name, h, size[, origin])	Create and return a 3D cubic lattice.
make_orthorombicp_lattice(name, hs, size[, ...])	Create and return a 3D orthorombic primitive lattice.

Implementation

class simphony.cuds.lattice.**Lattice**(*name*, *type*, *base_vect*, *size*, *origin*)
A Bravais lattice

Stores references to data containers (node related data).

name
str

name of the lattice

type
str

Bravais lattice type (should agree with the base_vect below).

base_vect
D x float

defines a Bravais lattice of dimension D = 2,3 (an alternative for primitive vectors).

size
tuple of D x int

number of lattice nodes in the direction of each axis.

origin
D x float

origin of lattice

base_vect

get_coordinate(*index*)
Get coordinate of the given index coordinate.

Parameters **index** (*D x int (node index coordinate)*) –

Returns

Return type D x float

get_node(*index*)
Get a copy of the node corresponding to the given index.

Parameters **index** (*tuple of D x int (node index coordinate)*) –

Returns

Return type A reference to a LatticeNode object

iter_nodes(*indices=None*)
Get an iterator over the LatticeNodes described by the indices.

Parameters **indices** (*iterable set of D x int, optional*) – node index coordinates

Returns

Return type A generator for LatticeNode objects

origin

size

type

update_node(*lat_node*)
Update the corresponding lattice node (data copied).

Parameters **lat_node** (*reference to a LatticeNode object*) – data copied from the given node

class simphony.cuds.lattice.**LatticeNode**(*index*, *data=None*)
A single node of a lattice.

index
tuple of D x int

D dimensional node index coordinate

data
DataContainer

simphony.cuds.lattice.**make_cubic_lattice**(*name*, *h*, *size*, *origin=(0, 0, 0)*)
Create and return a 3D cubic lattice.

Parameters

- **name** (*str*) –
- **h** (*float*) – lattice spacing.
- **size** (*3 x int*) – number of lattice nodes (in each axis direction).

- **origin** (*3 x float (default value = (0,0,0))*) – lattice origin.

Returns **lattice** – A reference to a Lattice object.

Return type *Lattice*

simphony.cuds.lattice.**make_hexagonal_lattice**(*name*, *h*, *size*, *origin=(0, 0)*)
Create and return a 2D hexagonal lattice.

Parameters

- **name** (*str*) –
- **h** (*float*) – lattice spacing.
- **size** (*2 x int*) – number of lattice nodes (in each axis direction).
- **origin** (*2 x float (default value = (0,0))*) – lattice origin.

Returns **lattice** – A reference to a Lattice object.

Return type *Lattice*

simphony.cuds.lattice.**make_orthorombicp_lattice**(*name*, *hs*, *size*, *origin=(0, 0, 0)*)
Create and return a 3D orthorombic primitive lattice.

Parameters

- **name** (*str*) –
- **hs** (*3 x float*) – lattice spacings (in each axis direction).
- **size** (*3 x int*) – number of lattice nodes (in each axis direction).
- **origin** (*3 x float (default value = (0,0,0))*) – lattice origin.

Returns **lattice** – A reference to a Lattice object.

Return type *Lattice*

simphony.cuds.lattice.**make_rectangular_lattice**(*name*, *hs*, *size*, *origin=(0, 0)*)
Create and return a 2D rectangular lattice.

Parameters

- **name** (*str*) –
- **hs** (*2 x float*) – lattice spacings (in each axis direction).
- **size** (*2 x int*) – number of lattice nodes (in each axis direction).
- **origin** (*2 x float (default value = (0,0))*) – lattice origin.

Returns **lattice** – A reference to a Lattice object.

Return type *Lattice*

simphony.cuds.lattice.**make_square_lattice**(*name*, *h*, *size*, *origin=(0, 0)*)
Create and return a 2D square lattice.

Parameters

- **name** (*str*) –
- **h** (*float*) – lattice spacing.
- **size** (*2 x int*) – number of lattice nodes (in each axis direction).
- **origin** (*2 x float (default value = (0,0))*) – lattice origin.

Returns **lattice** – A reference to a Lattice object.

Return type *Lattice*

Mesh module

This module contains the implementation to store, access, and modify a mesh

class simphony.cuds.mesh.**Cell**(*points*, *uid=None*, *data=None*)
Cell element

Element for storing 3D geometrical objects

Parameters

- **points** (*list of uid*) – list of points uids defining the cell.
- **uid** – uid of the cell.
- **data** (DataContainer) – object to store data relative to the cell

classmethod **from_cell**(*cell*)

class simphony.cuds.mesh.**Edge**(*points*, *uid=None*, *data=None*)
Edge element

Element for storing 1D geometrical objects

Parameters

- **points** (*list of uid*) – list of points uids defining the edge.
- **uid** – uid of the edge.
- **data** (DataContainer) – object to store data relative to the edge

classmethod **from_edge**(*edge*)

class simphony.cuds.mesh.**Element**(*points*, *uid=None*, *data=None*)
Element base class

Element for storing geometrical objects

Parameters

- **uid** – uid of the edge.
- **points** (*list of uid*) – list of points uids defining the edge.
- **data** (DataContainer) – object to store data relative to the element

points
list of uid

list of points uids defining the element.

uid
uuid of the element

data
DataContainer

Element data

classmethod **from_element**(*element*)

class simphony.cuds.mesh.**Face**(*points*, *uid=None*, *data=None*)
Face element

Element for storing 2D geometrical objects

Parameters

- **points** (*list of uid*) – list of points uids defining the face.
- **uid** – uid of the face.
- **data** (DataContainer) – object to store data relative to the face

classmethod from_face(*face*)

class simphony.cuds.mesh.**Mesh**(*name*)
Mesh object to store points and elements.

Stores general mesh information Points and Elements such as Edges, Faces and Cells and provide the methods to interact with them. The methods are divided in four different blocks:

1. methods to get the related item with the provided uuid;
2. methods to add a new item or replace;
3. generator methods that return iterators over all or some of the mesh items and;
4. inspection methods to identify if there are any edges, faces or cells described in the mesh.

Parameters **name** (*str*) – name of mesh

name
str

name of mesh

data
Data

Data relative to the mesh.

points
dictionary of Point

Points of the mesh.

edges
dictionary of Edge

Edges of the mesh.

faces
dictionary of Face

Faces of the mesh.

cells
dictionary of Cell

Cells of the mesh.

See also:

get_point, *get_edge*, *get_face*, *get_cell*, *add_point*, *add_edge*, *add_face*, *add_cell*, *update_point*, *update_edge*, *update_face*, *update_cell*, *iter_points*, *iter_edges*, *iter_faces*, *iter_cells*, *has_edges*, *has_faces*, *has_cells*

add_cell(*cell*)
Adds a new cell to the mesh.

Parameters **cell** (Cell) – Cell to be added to the mesh

Raises

- KeyError – If other cell with the same uid was already in the mesh
- TypeError – If the object provided is not a cell

add_edge(*edge*)
Adds a new edge to the mesh.

Parameters **edge** (Edge) – Edge to be added to the mesh

Raises

- KeyError – If other edge with the same uid was already in the mesh
- TypeError – If the object provided is not an edge

add_face(*face*)
Adds a new face to the mesh.

Parameters **face** (Face) – Face to be added to the mesh

Raises

- KeyError – If other face with the same uid was already in the mesh
- TypeError – If the object provided is not a face

add_point(*point*)
Adds a new point to the mesh.

Parameters **point** (Point) – Point to be added to the mesh

Raises

- KeyError – If other point with the same uid was already in the mesh
- TypeError – If the object provided is not a point

data

get_cell(*uid*)
Returns a cell with a given uid.

Returns the cell stored in the mesh identified by uuid . If such cell do not exists an exception is raised.

Parameters **uid** – uid of the desired cell.

Returns Cell with id identified by uid

Return type *Cell*

Raises Exception – If the cell identified by uuid was not found

get_edge(*uid*)
Returns an edge with a given uuid.

Returns the edge stored in the mesh identified by uuid. If such edge do not exists an exception is raised.

Parameters **uid** – uid of the desired edge.

Returns Edge identified by uid

Return type *Edge*

Raises `Exception` – If the edge identified by uuid was not found

get_face(*uid*)
Returns a face with a given uid.

Returns the face stored in the mesh identified by uid. If such face do not exists an exception is raised.

Parameters **uid** – uid of the desired face.

Returns Face identified by uid

Return type *Face*

Raises `Exception` – If the face identified by uuid was not found

get_point(*uid*)
Returns a point with a given uuid.

Returns the point stored in the mesh identified by uuid. If such point do not exists an exception is raised.

Parameters **uid** – uid of the desired point.

Returns Mesh point identified by uuid

Return type *Point*

Raises `Exception` – If the point identified by uuid was not found

has_cells()
Check if the mesh has cells

Returns True of there are cells inside the mesh, False otherwise

Return type bool

has_edges()
Check if the mesh has edges

Returns True of there are edges inside the mesh, False otherwise

Return type bool

has_faces()
Check if the mesh has faces

Returns True of there are faces inside the mesh, False otherwise

Return type bool

iter_cells(*cell_uids=None*)
Returns an iterator over the selected cells.

Returns an iterator over the cells with uid in cell_uids. If none of the uids in cell_uids exists, an empty iterator is returned. If there is no uuids inside cell_uuids, a iterator over all cells of the mesh is returned instead.

Parameters **cell_uids** (*list of uid, optional*) – uids of the desired cell, default empty

Returns Iterator over the selected cells

Return type iter

iter_edges(*edge_uids=None*)
Returns an iterator over the selected edges.

Returns an iterator over the edged with uid in edge_uid. If none of the uids in edge_uids exists, an empty iterator is returned. If there is no uids inside edge_uids, a iterator over all edges of the mesh is returned instead.

Parameters **edge_uids** (*list of uid, optional*) – uids of the desired edges, default empty

Returns Iterator over the selected edges

Return type iter

iter_faces(*face_uids=None*)
Returns an iterator over the selected faces.

Returns an iterator over the faces with uid in face_uids. If none of the uuids in face_uids exists, an empty iterator is returned. If there is no uids inside face_uids, a iterator over all faces of the mesh is returned instead.

Parameters **face_uids** (*list of uid, optional*) – uids of the desired faces, default empty

Returns Iterator over the selected faces

Return type iter

iter_points(*point_uids=None*)
Returns an iterator over the selected points.

Returns an iterator over the points with uid in point_uids. If none of the ids in point_uids exists, an empty iterator is returned. If there is no ids inside point_uids, a iterator over all points of the mesh is returned instead.

Parameters **point_uids** (*list of uid, optional*) – uids of the desired points, default empty

Returns Iterator over the selected points

Return type iter

update_cell(*cell*)
Updates the information of a cell.

Gets the mesh cell identified by the same uuid as the provided cell and updates its information with the one provided with the new cell.

Parameters **cell** (Cell) – Cell to be updated

Raises

- `KeyError` – If the cell was not found in the mesh
- `TypeError` – If the object provided is not a cell

update_edge(*edge*)
Updates the information of an edge.

Gets the mesh edge identified by the same id as the provided edge and updates its information with the one provided with the new edge.

Parameters **edge** (Edge) – Edge to be updated

Raises

- `KeyError` – If the edge was not found in the mesh
- `TypeError` – If the object provided is not an edge

update_face(*face*)
Updates the information of a face.

Gets the mesh face identified by the same uuid as the provided face and updates its information with the one provided with the new face.

Parameters **face** (Face) – Face to be updated

Raises

- `KeyError` – If the face was not found in the mesh
- `TypeError` – If the object provided is not a face

update_point(*point*)

Updates the information of a point.

Gets the mesh point identified by the same id as the provided point and updates its information with the one provided with the new point.

Parameters **point** (Point) – Point to be updated

Raises

- `KeyError` – If the point was not found in the mesh
- `TypeError` – If the object provided is not a point

class simphony.cuds.mesh.**Point**(*coordinates*, *uid=None*, *data=None*)

Coordinates describing a point in the space

Set of coordinates(x,y,z) describing a point in the space and data about that point

Parameters

- **uid** – uid of the point.
- **coordinates** (*list of double*) – set of coordinates (x,y,z) describing the point position.
- **data** (DataContainer) – object to store point data

uid

uuid of the point.

data

DataContainer

object to store point data

coordinates

list of double

set of coordinates (x,y,z) describing the point position.

classmethod from_point(*point*)

class simphony.cuds.particles.**Bond**(*particles*, *uid=None*, *data=None*)

Class reprensenting a bond.

uid

uuid.UUID

the unique id of the bond

particles

tuple

tuple of uuids of the particles that are participating in the bond.

data

DataContainer

DataContainer to store the attributes of the bond

classmethod from_bond(*bond*)

class simphony.cuds.particles.**Particle**(*coordinates=(0.0, 0.0, 0.0)*, *uid=None*, *data=None*)
Class representing a particle.

uid
uuid.UUID

the unique id of the particle

coordinates
list / tuple

x,y,z coordinates of the particle

data
DataContainer

DataContainer to store the attributes of the particle

classmethod from_particle(*particle*)

class simphony.cuds.particles.**Particles**(*name*)
Class that represents a container of particles and bonds.

Class provides methods to add particles and bonds, remove them and update them.

name
str

name of the particle container

_particles
dict

data structure for particles storage

_bonds
dict

data structure for bonds storage

data
DataContainer

data attributes of the element

add_bond(*bond*)
Adds the bond to the container.

Also like with particles, if the bond has an user defined id, it won't add the bond if a bond with the same id already exists, and if the bond has no id the particle container will generate an unique id. If the user wants to replace an existing bond in the container there is an 'update_bond' method for that purpose.

Parameters **new_bond** (Bond) – the new bond that will be included in the container.

Returns **uid** – The id of the added bond.

Return type uuid.UID

Raises **ValueError when the new particle already exists in the container.** –

See also:

update_bond(), *remove_bond()*

Examples

Having a Bond and a ParticleContainer just call the function passing the Bond as parameter.

```
>>> bond = Bond()
>>> part_container = Particles(name="foo")
>>> part_container.add_bond(bond)
```

add_particle(*particle*)

Adds the particle to the container.

If the new particle has no id, the particle container will generate a new unique id for it. If the particle has already an id (user set), it won't add the particle if a particle with the same id already exists. If the user wants to replace an existing particle in the container there is an 'update_particle' method for that purpose.

Parameters **particle** (Particle) – the new particle that will be included in the container.

Returns **uid** – The id of the added particle.

Return type uuid.UUID

Raises **ValueError when the new particle already exists in the container.** –

See also:

update_particle(), *remove_particle()*

Examples

Having a Particle and a ParticleContainer just call the function passing the Particle as parameter.

```
>>> part = Particle()
>>> part_container = Particles(name="foo")
>>> part_container.add_particle(part)
```

data

get_bond(*uid*)

Returns a copy of the bond with the 'bond_id' id.

Parameters **uid** (*uuid.UUID*) – the id of the bond

Raises **KeyError when the bond is not in the container.** –

Returns

Return type A copy of the bond

get_particle(*uid*)

Returns a copy of the particle with the 'particle_id' id.

Parameters **uid** (*uuid.UUID*) – the id of the particle

Raises **KeyError when the particle is not in the container.** –

Returns

Return type A copy of the particle

has_bond(*uid*)

Checks if a bond with the given id already exists in the container.

has_particle(*uid*)

Checks if a particle with the given id already exists in the container.

iter_bonds(*ids=None*)

Generator method for iterating over the bonds of the container.

It can recieve any kind of sequence of bond ids to iterate over those concrete bond. If nothing is passed as parameter, it will iterate over all the bonds.

Parameters **bond_ids** (*array_like*) – sequence containing the id's of the bond that will be iterated.

Yields *Yields each bond to be used.*

Raises

- **KeyError exception if any of the ids passed as parameters are not**
- **in the container.** –

See also:

iter_particles(), *add_bond()*, *remove_bond()*, *update_bond()*

Examples

It can be used with a sequence as parameter or withouth it:

```
>>> part_container = Particles(name="foo")
>>> ...
>>> for bond in part_container.iter_bonds([id1, id2, id3]):
        ...  #do stuff
        #take the bond back to the container so it will be updated
        #in case we need it
        part_container.update_bond(bond)
```

```
>>> for bond in part_container.iter_bond():
        ...  #do stuff; it will iterate over all the bond
        #take the bond back to the container so it will be updated
        #in case we need it
        part_container.update_bond(bond)
```

iter_particles(*ids=None*)

Generator method for iterating over the particles of the container.

It can recieve any kind of sequence of particle ids to iterate over those concrete particles. If nothing is passed as parameter, it will iterate over all the particles.

Parameters **ids** (*iterable*) – sequence containing the id's of the particles that will be iterated.

Yields **particle** (*Particle*) – Yields each particle to be used.

Raises

- **KeyError exception if any of the ids passed as parameters are not**
- **in the container.** –

See also:

iter_bonds(), *add_particle()*, *remove_particle()*, *update_particle()*

Examples

It can be used with a sequence as parameter or withouth it:

```
>>> part_container = Particles(name="foo")
>>> ...
>>> for particle in part_container.iter_particles([id1, id2, id3]):
        ...  #do stuff
        #take the particle back to the container so it will be updated
        #in case we need it
        part_container.update_particle(particle)
```

```
>>> for particle in part_container.iter_particles():
        ...  #do stuff; it will iterate over all the particles
        #take the particle back to the container so it will be updated
        #in case we need it
        part_container.update_particle(particle)
```

remove_bond(*uid*)

Removes the bond with the uid from the container.

The id passed as parameter should exists in the container. If it doesn't exists, nothing will happen.

Parameters **uid** (*uuid.UUID*) – the id of the bond to be removed.

See also:

add_bond(), *update_bond()*

Examples

Having an id of an existing bond, pass it to the function.

```
>>> part_container = Particles(name="foo")
>>> ...
>>> bond = part_container.get_bond(id)
>>> ...
>>> part_container.remove_bond(bond.uid)
or directly
>>> part_container.remove_bond(id)
```

remove_particle(*uid*)

Removes the particle with uid from the container.

The id passed as parameter should exists in the container. Otherwise an expcetion will be raised.

Parameters **uid** (*uuid.UUID*) – the id of the particle to be removed.

Raises **KeyError exception if the particle doesn't exist.** –

See also:

add_particle(), *update_particle()*

Examples

Having an id of an existing particle, pass it to the function.

```
>>> part_container = Particles(name="foo")
>>> ...
>>> part = part_container.get_particle(uid)
>>> ...
>>> part_container.remove_particle(part.uid)
or directly
>>> part_container.remove_particle(uid)
```

update_bond(*bond*)

Replaces an existing bond.

Takes the id of 'bond' and searchs inside the container for that bond. If the bond exists, it is replaced with the new bond passed as parameter. If the bond doesn't exist, it will raise an exception.

Parameters **bond** (Bond) – the bond that will be replaced.

Raises **ValueError exception if the bond doesn't exists.** –

See also:

add_bond(), *remove_bond()*

Examples

Having a Bond that already exists in the container (taken with the 'get_bond' method for example) and a Particles just call the function passing the Bond as parameter.

```
>>> part_container = Particles(name="foo")
>>> ...
>>> bond = part_container.get_bond(uid)
>>> ... #do whatever you want with the bond
>>> part_container.update_bond(bond)
```

update_particle(*particle*)

Replaces an existing particle.

Takes the id of 'particle' and searchs inside the container for that particle. If the particle exists, it is replaced with the new particle passed as parameter. If the particle doesn't exist, it will raise an exception.

Parameters **particle** (Particle) – the particle that will be replaced.

Raises **ValueError exception if the particle does not exists.** –

See also:

add_particle(), *remove_particle()*

Examples

Having a Particle that already exists in the container (taken with the 'get_particle' method for example) and a ParticleContainer just call the function passing the Particle as parameter.

```
>>> part_container = Particles(name="foo")
>>> ...
>>> part = part_container.get_particle(uid)
>>> ... #do whatever you want with the particle
>>> part_container.update_particle(part)
```

3.3 HDF5 IO

The CUDS to HDF5 file adapters.

Classes

H5CUDS(handle)	Access to CUDS-hdf5 formatted files.
DataContainerTable(root[, name, record])	A proxy class to an HDF5 group node with serialised DataContainers.
IndexedDataContainerTable(root[, name, ...])	A proxy class to an HDF5 group node with serialised DataContainers.
H5Particles(group)	An HDF5 backed particle container.
H5CUDSItems(root, record[, name])	A proxy class to an HDF5 group node with serialised CUDS items.

Table descriptions

Implementation

class simphony.io.h5_cuds.**H5CUDS**(*handle*)

Bases: object

Access to CUDS-hdf5 formatted files.

add_lattice(*lattice*)

Add lattice to the file.

Parameters

- **lattice** (Lattice) – lattice to be added
- **Returns** –
- **----------** –
- **H5Lattice** – The lattice newly added to the file.

add_mesh(*mesh*)

Add a mesh to the file.

Parameters

- **name** (*str*) – name of the mesh
- **mesh_container** (*ABCMesh, optional*) – mesh to be added. If none is give, then an empty mesh is added.
- **Returns** –
- **----------** –
- **H5Mesh** – The mesh newly added to the file. See get_mesh for more information.

add_particles(*particles*)

Add particle container to the file.

Parameters **particles** (ABCParticles) – Particle container to be added.

Returns **particles** – A newly created container proxying the data in the HDF5 file.

Return type *H5Particles*

close()
Closes a file

delete_lattice(*name*)
Delete lattice from file.

Parameters **name** (*str*) – name of lattice to delete

delete_mesh(*name*)
Delete mesh from file.

Parameters **name** (*str*) – name of the mesh to delete

delete_particles(*name*)
Delete particle container from file.

Parameters **name** (*str*) – name of particle container to delete

get_lattice(*name*)
Get lattice from file.

The returned lattice can be used to query and change the related data stored in the file. If the file has been closed then the lattice should no longer be used.

Parameters **name** (*str*) – name of lattice to return

get_mesh(*name*)
Get mesh from file.

The returned mesh can be used to query and change the related data stored in the file. If the file has been closed then the mesh should no longer be used.

Parameters **name** (*str*) – name of the mesh to return

get_particles(*name*)
Get particle container from file. The returned particle container can be used to query and change the related data stored in the file. If the file has been closed then the particle container should no longer be used.

Parameters **name** (*str*) – name of particle container to return

iter_lattices(*names=None*)
Returns an iterator over a subset or all of the lattices.

Parameters **names** (*list of str*) – names of specific lattices to be iterated over. If names is not given, then all lattices will be iterated over.

iter_meshes(*names=None*)
Returns an iterator over a subset or all of the meshes.

Parameters **names** (*list of str*) – names of specific meshes to be iterated over. If names is not given, then all meshes will be iterated over.

iter_particles(*names=None*)
Returns an iterator over a subset or all of the particle containers. :param names: names of specific particle containers to be iterated over.

If names is not given, then all particle containers will be iterated over.

classmethod open(*filename*, *mode='a'*, *title=''*)
Returns a SimPhony file and returns an opened CudsFile

Parameters

- **filename** (*str*) – Name of file to be opened.
- **mode** (*str*) – The mode to open the file:
  - `w` – Write; a new file is created (an existing file with the same name would be deleted).
  - `a` – Append; an existing file is opened for reading and writing, and if the file does not exist it is created.
  - `r` – ReadOnly; This is a very restrictive mode that will through errors at any attempt to modify the data.
- **title** (*str*) – Title attribute of root node (only applies to a file which is being created)

valid()
Checks if file is valid (i.e. open)

class simphony.io.h5_particles.**H5BondItems**(*root*, *name='bonds'*)
Bases: *simphony.io.h5_cuds_items.H5CUDSItems*

A proxy class to an HDF5 group node with serialised Bonds

The class implements the Mutable-Mapping api where each Bond instance is mapped to uuid.

class simphony.io.h5_particles.**H5ParticleItems**(*root*, *name='particles'*)
Bases: *simphony.io.h5_cuds_items.H5CUDSItems*

A proxy class to an HDF5 group node with serialised Particles

The class implements the Mutable-Mapping api where each Particle instance is mapped to uuid.

class simphony.io.h5_particles.**H5Particles**(*group*)
Bases: *simphony.cuds.abstractparticles.ABCParticles*

An HDF5 backed particle container.

add_bond(*bond*)
Add bond

If bond has an id then this is used. If the bond's id is None then a id is generated for the bond.

Returns id of bond

Return type int

Raises `ValueError` – if an id is given which already exists.

add_particle(*particle*)
Add particle

If particle has a uid set then this is used. If the particle's uid is None then a new uid is generated for the particle.

Returns **uid** – uid of particle.

Return type uuid.UUID

Raises `ValueError` – The particle uid already exists in the container.

data

get_bond(*uid*)

get_particle(*uid*)

has_bond(*uid*)
Checks if a bond with id "id" exists in the container.

has_particle(*uid*)
Checks if a particle with id "id" exists in the container.

iter_bonds(*ids=None*)
Get iterator over particles

iter_particles(*ids=None*)
Get iterator over particles

name
The name of the container

remove_bond(*uid*)

remove_particle(*uid*)

update_bond(*bond*)

update_particle(*particle*)

class simphony.io.h5_cuds_items.**H5CUDSItems**(*root*, *record*, *name='items'*)
Bases: _abcoll.MutableMapping

A proxy class to an HDF5 group node with serialised CUDS items.

The class implements the Mutable-Mapping api where each item instance is mapped to uuid.

__delitem__(*uid*)
Delete the row.

__getitem__(*uid*)
Return the Particle with the provided id.

__iter__()
Iterate over all the rows

__len__()
The number of rows in the table.

A runtime error is raised when the number of rows in the items and data tables is not equal.

__setitem__(*uid*, *item*)
Set the particle in row with item.

- If the uid does not exist in the Table a new row will be appended.
- If the `item.uid` is `None` then `item.uid = uid`.

add_safe(*item*)
Add item while checking for a unique uid.

Note: The item is expected to already have a uid set.

add_unsafe(*item*)
Add item without checking for a unique uid.

Note: The item is expected to already have a uid set.

itersequence(*sequence*)
Iterate over a sequence of row ids.

update_existing(*item*)
Update an item if it already exists.

valid
A PyTables table is opened/created and the object is valid.

class simphony.io.data_container_table.**DataContainerTable**(*root*, *name='data_containers'*, *record=None*)

Bases: _abcoll.MutableMapping

A proxy class to an HDF5 group node with serialised DataContainers.

The class implements the Mutable-Mapping api where each DataContainer instance is mapped to uuid.

__delitem__(*uid*)
Delete the row.

__getitem__(*uid*)
Return the DataContainer in row.

__iter__()
Iterate over all the rows

__len__()
The number of rows in the table.

__setitem__(*uid*, *data*)
Set the data in row from the DataContainer.

append(*data*)
Append the data to the end of the table.

Parameters **data** (DataContainer) – The DataContainer instance to save.

Returns **uid** – The index of the saved row.

Return type uuid.UUID

itersequence(*sequence*)
Iterate over a sequence of row ids.

valid
A PyTables table is opened/created and the object is valid.

class simphony.io.indexed_data_container_table.**IndexedDataContainerTable**(*root*, *name='data_containers'*, *record=None*, *expected_number=None*)

Bases: _abcoll.Sequence

A proxy class to an HDF5 group node with serialised DataContainers.

The class implements the Sequence api where each DataContainer instance is mapped to the row. In addition the class implements update (i.e. __setitem__) and append.

__getitem__(*index*)
Return the DataContainer in index.

__iter__()
Iterate over all the rows

__len__()
The number of rows in the table.

__setitem__(*index*, *data*)
Update the data in index.

append(*data*)
Append the data to the end of the table.

Parameters **data** (DataContainer) – The DataContainer instance to save.

Returns **index** – The index of the saved row.

Return type int

valid

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

S

simphony.core.data_container, 5
simphony.cuds.abstractlattice, 7
simphony.cuds.abstractmesh, 6
simphony.cuds.abstractparticles, 6
simphony.cuds.lattice, 8
simphony.cuds.mesh, 11
simphony.cuds.particles, 16
simphony.io.data_container_description, 27
simphony.io.data_container_table, 26
simphony.io.h5_cuds, 22
simphony.io.h5_cuds_items, 25
simphony.io.h5_particles, 24
simphony.io.indexed_data_container_table, 26

Symbols

__delitem__() (simphony.io.data_container_table.DataContainerTable method), 26
__delitem__() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
__getitem__() (simphony.io.data_container_table.DataContainerTable method), 26
__getitem__() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
__getitem__() (simphony.io.indexed_data_container_table.IndexedDataContainerTable method), 26
__iter__() (simphony.io.data_container_table.DataContainerTable method), 26
__iter__() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
__iter__() (simphony.io.indexed_data_container_table.IndexedDataContainerTable method), 26
__len__() (simphony.io.data_container_table.DataContainerTable method), 26
__len__() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
__len__() (simphony.io.indexed_data_container_table.IndexedDataContainerTable method), 26
__setitem__() (simphony.core.data_container.DataContainer method), 5
__setitem__() (simphony.io.data_container_table.DataContainerTable method), 26
__setitem__() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
__setitem__() (simphony.io.indexed_data_container_table.IndexedDataContainerTable method), 26
_bonds (simphony.cuds.particles.Particles attribute), 17
_particles (simphony.cuds.particles.Particles attribute), 17

A

ABCLattice (class in simphony.cuds.abstractlattice), 7
ABCMesh (class in simphony.cuds.abstractmesh), 6
ABCParticles (class in simphony.cuds.abstractparticles), 6
add_bond() (simphony.cuds.abstractparticles.ABCParticles method), 6
add_bond() (simphony.cuds.particles.Particles method), 17
add_bond() (simphony.io.h5_particles.H5Particles method), 24
add_cell() (simphony.cuds.abstractmesh.ABCMesh method), 6
add_cell() (simphony.cuds.mesh.Mesh method), 12
add_edge() (simphony.cuds.abstractmesh.ABCMesh method), 6
add_edge() (simphony.cuds.mesh.Mesh method), 13
add_face() (simphony.cuds.abstractmesh.ABCMesh method), 6
add_face() (simphony.cuds.mesh.Mesh method), 13
add_lattice() (simphony.io.h5_cuds.H5CUDS method), 22
add_mesh() (simphony.io.h5_cuds.H5CUDS method), 22
add_particle() (simphony.cuds.abstractparticles.ABCParticles method), 6
add_particle() (simphony.cuds.particles.Particles method), 18
add_particle() (simphony.io.h5_particles.H5Particles method), 24
add_particles() (simphony.io.h5_cuds.H5CUDS method), 22
add_point() (simphony.cuds.abstractmesh.ABCMesh method), 6
add_point() (simphony.cuds.mesh.Mesh method), 13
add_safe() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
add_unsafe() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
append() (simphony.io.data_container_table.DataContainerTable method), 26
append() (simphony.io.indexed_data_container_table.IndexedDataContainer method), 27

B

base_vect (simphony.cuds.abstractlattice.ABCLattice attribute), 7

base_vect (simphony.cuds.lattice.Lattice attribute), 8, 9
Bond (class in simphony.cuds.particles), 16

C

Cell (class in simphony.cuds.mesh), 11
cells (simphony.cuds.mesh.Mesh attribute), 12
close() (simphony.io.h5_cuds.H5CUDS method), 23
coordinates (simphony.cuds.mesh.Point attribute), 16
coordinates (simphony.cuds.particles.Particle attribute), 17

D

data (simphony.cuds.lattice.LatticeNode attribute), 9
data (simphony.cuds.mesh.Element attribute), 11
data (simphony.cuds.mesh.Mesh attribute), 12, 13
data (simphony.cuds.mesh.Point attribute), 16
data (simphony.cuds.particles.Bond attribute), 16
data (simphony.cuds.particles.Particle attribute), 17
data (simphony.cuds.particles.Particles attribute), 17, 18
data (simphony.io.h5_particles.H5Particles attribute), 24
DataContainer (class in simphony.core.data_container), 5
DataContainerTable (class in simphony.io.data_container_table), 26
delete_lattice() (simphony.io.h5_cuds.H5CUDS method), 23
delete_mesh() (simphony.io.h5_cuds.H5CUDS method), 23
delete_particles() (simphony.io.h5_cuds.H5CUDS method), 23

E

Edge (class in simphony.cuds.mesh), 11
edges (simphony.cuds.mesh.Mesh attribute), 12
Element (class in simphony.cuds.mesh), 11

F

Face (class in simphony.cuds.mesh), 11
faces (simphony.cuds.mesh.Mesh attribute), 12
from_bond() (simphony.cuds.particles.Bond class method), 16
from_cell() (simphony.cuds.mesh.Cell class method), 11
from_edge() (simphony.cuds.mesh.Edge class method), 11
from_element() (simphony.cuds.mesh.Element class method), 11
from_face() (simphony.cuds.mesh.Face class method), 12
from_particle() (simphony.cuds.particles.Particle class method), 17
from_point() (simphony.cuds.mesh.Point class method), 16

G

get_bond() (simphony.cuds.abstractparticles.ABCParticles method), 6
get_bond() (simphony.cuds.particles.Particles method), 18
get_bond() (simphony.io.h5_particles.H5Particles method), 24
get_cell() (simphony.cuds.abstractmesh.ABCMesh method), 6
get_cell() (simphony.cuds.mesh.Mesh method), 13
get_coordinate() (simphony.cuds.abstractlattice.ABCLattice method), 7
get_coordinate() (simphony.cuds.lattice.Lattice method), 9
get_edge() (simphony.cuds.abstractmesh.ABCMesh method), 6
get_edge() (simphony.cuds.mesh.Mesh method), 13
get_face() (simphony.cuds.abstractmesh.ABCMesh method), 6
get_face() (simphony.cuds.mesh.Mesh method), 14
get_lattice() (simphony.io.h5_cuds.H5CUDS method), 23
get_mesh() (simphony.io.h5_cuds.H5CUDS method), 23
get_node() (simphony.cuds.abstractlattice.ABCLattice method), 7
get_node() (simphony.cuds.lattice.Lattice method), 9
get_particle() (simphony.cuds.abstractparticles.ABCParticles method), 6
get_particle() (simphony.cuds.particles.Particles method), 18
get_particle() (simphony.io.h5_particles.H5Particles method), 24
get_particles() (simphony.io.h5_cuds.H5CUDS method), 23
get_point() (simphony.cuds.abstractmesh.ABCMesh method), 6
get_point() (simphony.cuds.mesh.Mesh method), 14

H

H5BondItems (class in simphony.io.h5_particles), 24
H5CUDS (class in simphony.io.h5_cuds), 22
H5CUDSItems (class in simphony.io.h5_cuds_items), 25
H5ParticleItems (class in simphony.io.h5_particles), 24
H5Particles (class in simphony.io.h5_particles), 24
has_bond() (simphony.cuds.abstractparticles.ABCParticles method), 7
has_bond() (simphony.cuds.particles.Particles method), 18
has_bond() (simphony.io.h5_particles.H5Particles method), 24
has_cells() (simphony.cuds.abstractmesh.ABCMesh method), 6
has_cells() (simphony.cuds.mesh.Mesh method), 14
has_edges() (simphony.cuds.abstractmesh.ABCMesh method), 6
has_edges() (simphony.cuds.mesh.Mesh method), 14
has_faces() (simphony.cuds.abstractmesh.ABCMesh method), 6

has_faces() (simphony.cuds.mesh.Mesh method), 14
has_particle() (simphony.cuds.abstractparticles.ABCParticles method), 7
has_particle() (simphony.cuds.particles.Particles method), 18
has_particle() (simphony.io.h5_particles.H5Particles method), 25

I

index (simphony.cuds.lattice.LatticeNode attribute), 9
IndexedDataContainerTable (class in simphony.io.indexed_data_container_table), 26
iter_bonds() (simphony.cuds.abstractparticles.ABCParticles method), 7
iter_bonds() (simphony.cuds.particles.Particles method), 19
iter_bonds() (simphony.io.h5_particles.H5Particles method), 25
iter_cells() (simphony.cuds.abstractmesh.ABCMesh method), 6
iter_cells() (simphony.cuds.mesh.Mesh method), 14
iter_edges() (simphony.cuds.abstractmesh.ABCMesh method), 6
iter_edges() (simphony.cuds.mesh.Mesh method), 14
iter_faces() (simphony.cuds.abstractmesh.ABCMesh method), 6
iter_faces() (simphony.cuds.mesh.Mesh method), 15
iter_lattices() (simphony.io.h5_cuds.H5CUDS method), 23
iter_meshes() (simphony.io.h5_cuds.H5CUDS method), 23
iter_nodes() (simphony.cuds.abstractlattice.ABCLattice method), 7
iter_nodes() (simphony.cuds.lattice.Lattice method), 9
iter_particles() (simphony.cuds.abstractparticles.ABCParticles method), 7
iter_particles() (simphony.cuds.particles.Particles method), 19
iter_particles() (simphony.io.h5_cuds.H5CUDS method), 23
iter_particles() (simphony.io.h5_particles.H5Particles method), 25
iter_points() (simphony.cuds.abstractmesh.ABCMesh method), 6
iter_points() (simphony.cuds.mesh.Mesh method), 15
itersequence() (simphony.io.data_container_table.DataContainerTable method), 26
itersequence() (simphony.io.h5_cuds_items.H5CUDSItems method), 25

L

Lattice (class in simphony.cuds.lattice), 8
LatticeNode (class in simphony.cuds.lattice), 9

M

make_cubic_lattice() (in module simphony.cuds.lattice), 9
make_hexagonal_lattice() (in module simphony.cuds.lattice), 10
make_orthorombicp_lattice() (in module simphony.cuds.lattice), 10
make_rectangular_lattice() (in module simphony.cuds.lattice), 10
make_square_lattice() (in module simphony.cuds.lattice), 10
Mesh (class in simphony.cuds.mesh), 12

N

name (simphony.cuds.abstractlattice.ABCLattice attribute), 7
name (simphony.cuds.abstractmesh.ABCMesh attribute), 6
name (simphony.cuds.abstractparticles.ABCParticles attribute), 6
name (simphony.cuds.lattice.Lattice attribute), 8
name (simphony.cuds.mesh.Mesh attribute), 12
name (simphony.cuds.particles.Particles attribute), 17
name (simphony.io.h5_particles.H5Particles attribute), 25

O

open() (simphony.io.h5_cuds.H5CUDS class method), 23
origin (simphony.cuds.abstractlattice.ABCLattice attribute), 7
origin (simphony.cuds.lattice.Lattice attribute), 9

P

Particle (class in simphony.cuds.particles), 16
Particles (class in simphony.cuds.particles), 17
particles (simphony.cuds.particles.Bond attribute), 16
Point (class in simphony.cuds.mesh), 16
points (simphony.cuds.mesh.Element attribute), 11
points (simphony.cuds.mesh.Mesh attribute), 12

R

remove_bond() (simphony.cuds.abstractparticles.ABCParticles method), 7
remove_bond() (simphony.cuds.particles.Particles method), 20
remove_bond() (simphony.io.h5_particles.H5Particles method), 25
remove_particle() (simphony.cuds.abstractparticles.ABCParticles method), 7
remove_particle() (simphony.cuds.particles.Particles method), 20
remove_particle() (simphony.io.h5_particles.H5Particles method), 25

S

simphony.core.data_container (module), 5
simphony.cuds.abstractlattice (module), 7
simphony.cuds.abstractmesh (module), 6
simphony.cuds.abstractparticles (module), 6
simphony.cuds.lattice (module), 8
simphony.cuds.mesh (module), 11
simphony.cuds.particles (module), 16
simphony.io.data_container_description (module), 27
simphony.io.data_container_table (module), 26
simphony.io.h5_cuds (module), 22
simphony.io.h5_cuds_items (module), 25
simphony.io.h5_particles (module), 24
simphony.io.indexed_data_container_table (module), 26
size (simphony.cuds.abstractlattice.ABCLattice attribute), 7
size (simphony.cuds.lattice.Lattice attribute), 8, 9

T

type (simphony.cuds.abstractlattice.ABCLattice attribute), 7
type (simphony.cuds.lattice.Lattice attribute), 8, 9

U

uid (simphony.cuds.mesh.Element attribute), 11
uid (simphony.cuds.mesh.Point attribute), 16
uid (simphony.cuds.particles.Bond attribute), 16
uid (simphony.cuds.particles.Particle attribute), 17
update() (simphony.core.data_container.DataContainer method), 5
update_bond() (simphony.cuds.abstractparticles.ABCParticles method), 7
update_bond() (simphony.cuds.particles.Particles method), 21
update_bond() (simphony.io.h5_particles.H5Particles method), 25
update_cell() (simphony.cuds.abstractmesh.ABCMesh method), 6
update_cell() (simphony.cuds.mesh.Mesh method), 15
update_edge() (simphony.cuds.abstractmesh.ABCMesh method), 6
update_edge() (simphony.cuds.mesh.Mesh method), 15
update_existing() (simphony.io.h5_cuds_items.H5CUDSItems method), 25
update_face() (simphony.cuds.abstractmesh.ABCMesh method), 6
update_face() (simphony.cuds.mesh.Mesh method), 15
update_node() (simphony.cuds.abstractlattice.ABCLattice method), 8
update_node() (simphony.cuds.lattice.Lattice method), 9
update_particle() (simphony.cuds.abstractparticles.ABCParticles method), 7
update_particle() (simphony.cuds.particles.Particles method), 21
update_particle() (simphony.io.h5_particles.H5Particles method), 25
update_point() (simphony.cuds.abstractmesh.ABCMesh method), 6
update_point() (simphony.cuds.mesh.Mesh method), 16

V

valid (simphony.io.data_container_table.DataContainerTable attribute), 26
valid (simphony.io.h5_cuds_items.H5CUDSItems attribute), 26
valid (simphony.io.indexed_data_container_table.IndexedDataContainerTab attribute), 27
valid() (simphony.io.h5_cuds.H5CUDS method), 24